Line 1 specifies the match condition: this template will match any element that contains a wdktags:attachTo subelement. Section 2 contains XSL logic for determining what root element should be used as the starting point for the value of the path attribute. If the developer specifies a root attribute, then the value of that attribute is used, otherwise the root element defaults to the wdk:model node of the model page. Section 3 invokes the getNodes ( ) method on the WDKDomUtils class. That method returns the set of nodes that can be accessed from the root node through the path given in the path attribute of the wdktags:attachTo directive. Section 4 checks for error conditions and sets up the iteration through the set of DOM elements returned in section 3. In section 5 the current xsp node (the value of the xspCurrentNode variable) is saved on a stack, and its value is replaced with the next node from the set of nodes returned in section 3. Since the XSP processor uses the xspCurrentNode variable to mark the current "insertion point"—i.e. the location where the next DOM node will be inserted in the Document, this operation effectively copies the current subtree (the widget) to each node returned in section 3. (Sections 6 and 7 perform the actual copying.) Finally, section 8 restores the value of the xspCurrentNode and resumes the iteration.

The following section describes the implementation of the nodeRef tag.

```
1    <xsl:template match="wdktags:nodeRef">
2         <xsl:variable name="root">
               <xsl:choose>
                    <xsl:when test="@source"><xsl:value-of-select="@source"/></xsl:when>
                    <xsl:otherwise>wdkwidgetNode</xsl:otherwise>
               </xsl:choose>
          </xsl:variable>
3         <xsp:logic>{
               Element wdkChildNode = WDKDomUtils.getChildNode((Element)<xsl:value-of
     select="$root"/>,"<xsl:value-of select="path"/>");
               <xsp:content><xsp:expr>WDKDomUtils.getTextValue(wdkChildNode)</xsp:expr></xsp:content>
               }
          </xsp:logic>
     </xsl:template>
```

Line 1 specifies the match condition: this rule matches every nodeRef tag. Section 2 determines the root node: if the source attribute is given then the value of that attribute is used, otherwise the value of wdkwidgetNode Java variable is used. The wdkwidgetNode variable is initialized in the wdktags:attachTo template described above. This way, if nodeRef is used in the context of an attachTo tag, the root node is the same node the widget is copied to. The actual node whose value is needed is located by following the path from the root node. Finally, the text value of the node is computed by calling the WDKDomUtils.getTextValue ( ) method.

Structure of View Pages

View pages are XSLT stylesheets. The role of the view stylesheet is to convert the XML document produced by executing the model file (and the subsequent widget transformation) to a format understood by the user agent. For example, for desktop browsers this typically means conversion to an HTML representation. Since model pages have a well-defined structure, view pages are also highly regular. For example, there are a number of model page elements that should not be rendered (such as wdk:head element and its content should not be copied to the output). Other model pages nodes have a standard representation in HTML (or in the desired output format). For example, the rule for rendering wdk:page is to generate the <html> element, the <head> element containing the <title> element. These common templates are all grouped in a default stylesheet that can be imported using the <xsl:import> directive by every view page. As a result, for simple pages, the view page needs to contain a singe cusomized xsl:template rule that matches on the "wdk:model" node. This template is responsible for rendering the data as well as the widgets.

Example: default view transformation templates

```
1    <?xml version="1.0"?>
     <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/
     1999/XSL/Transform"
     xmlns:wdk="http://www.saba.com/XML/WDK">
     <xsl:output method="xml" indent="yes"/>
     <xsl:strip-space elements="*"/>
2    <xsl:template match"/">
          <xsl:variable name="titleLabel"><xsl:value-of select=
          "//wdk:head/wdk:title"/></xsl:variable>
          <html>
               <head>
                    <title><xsl:value-of select="$titleLabel"/></title>
               </head>
               <body>
                    <xsl:apply-templates/>
```

-continued

```
               </body>
          </html>
     </xsl:template>
3    <xsl:template match="* | @*|text()|comment()" priority="-1">
          <xsl:copy>
               <xsl:apply-templates select="* | @*|text()|comment()"/>
          </xsl:copy>
     </xsl:template>
4    <!-- eliminate the wdk:head element and all children of
     wdk:widgets -->
     <xsl:template match="wdk:head | wdk:widgets">
     </xsl:template>
5    <!-- replace widget with span (so we can do CSS on it) and process
     their children -->
     <xsl:template match="wdk:widget">
          <span class="{@name}">
               <xsl:apply-templates/>
          </span>
          <br/>
     </xsl:template>
6    <xsl:template match="wdk:page">
          <xsl:apply-templates/>
     </xsl:template>
     </xsl:stylesheet>
```

Section 1 defines the namespaces used in the stylesheet. Section 2 defines the root level template. This template produces the html tags, and generates the html head element complete with the title element. Section 3 defines the default template: every element, attribute, text and comment is copied to the resulting document, unless a more specific template provides different instructions. Section 4 specifies a template for eliminating the wdk:head and wdk:widgets elements and their contents (since the contents of these tags should not be rendered using the default template defined in section 3). Section 5 introduces a template for transforming every widget by wrapping them into a span element replacing the wdk:widget "wrapper". This makes it possible to use CSS styling on a per named-widget basis. Finally, section 6 defines the template for processing the wdk:page element.

A View Page Example

```
1    <?xml version="1.0"?>
     <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/
     1999/XSL/Transform"
     xmlns:wdk="http://www.saba.com/XML/WDK">
2    <xsl:import href="../xsl/view/wdk_defaultview.xsl"/>
3    <xsl:template match="wdk:model">
4         <h2 align="center"><xsl:value-of
          select="/wdk:page/wdk:head/wdk:title"/></h2>
5         <p>
          <xsl:value-of select="/wdk:page/wdk:head/wdk:labels/wdk:label
          [@name='nameLabel']"/>
6         <xsl:for-each select="parents/parent">
               <xsl:value-of select="name"/>
               <xsl:text> &gt; </xsl:text>
          </xsl:for-each>
          <xsl:value-of select="parents/leaf/name"/>
          </p>
```

-continued

```
7         <xsl:apply-templates select="//wdk:widget"/>
8    </xsl:template>
     </xsl:stylesheet>
```

Section 2 imports the stylesheet containing the default templates. Line 3 defines the rule for processing the wdk:model node. Line 4 displays the title of the page by accessing the wdk:title tag inside the wdk:head tag. Section 6 iterates through each "parent" element inside the wdk:model element and displays its name. In section 7 any widget produced by the model page is displayed.

The wdk taglibrary

The wdk taglibrary contains a number of tags to simplify the development wdk model pages. The tag library includes tags for:

handling resource bundles for page internationalization,

invoking commands to generate XML representation of the data retrieved from the database,

managing the connectivity between widgets and the produced data model,

managing the input and output parameters to the model page,

etc.

To make the tag library accessible by the processing engine, the following line is inserted in cocoon.properties:

processor.xsp.logicsheet.wdktags.java=s:/sys/java/web/com/saba/web/xsl/taglib/wdk_taglib.xsl

The value of the above property identifies the location of the taglibrary stylesheet. The taglibrary stylesheet contains a number of xsl:import directives to import templates responsible for implementing subsets of tags and it also contains a number of default templates, as the code example below shows:

```
<?xml version"1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsp="http://www.apache.org/1999/XSP/Core"
xmlns:wdktags="http://www.saba.com/XML/WDK/taglib"
xmlns:wdk="http://www.saba.com/XML/WDK">
<xsl:preserve-space elements="*"/>
<xsl:include href="wdk_param.xsl"/>
<xsl:include href="wdk_i18n.xsl"/>
<xsl:include href="wdk_command.xsl"/>
<xsl:include href="wdk_control.xsl"/>
<xsl:include href="wdk_site.xsl"/>
<xsl:template match="xsp:page">
 <xsl:copy>
   <! — need to explicitly call some logic in the wdk_command stylesheet ->
        <xsl:call-template name="command_header"/>
        <! — need to explicitly call some logic in the control stylesheet ->
        <xsl:call-template name="control_header"/>
   <xsl:apply-templates/>
 </xsl:copy>
</xsl:template>
<xsl:template match="@*|*|test( )|processing-instruction( )|comment( )" priority="-1">
 <xsl:copy>
        <xsl:apply-templates select="@*|*|text( )|processing-instruction( )|comment( )"/>
 </xsl:copy>
</xsl:template>
<xsl:template match="wdk:head">
 <xsl:copy>
  <wdk:site>
     <href>/<xsp:expr>wdkRoot</xsp:expr>/</href>
     <imageRoot><xsp:expr>wdkSite.getImageRoot( )</xsp:expr></imageRoot>
```